AMENDMENT TO THE SPECIFICATION:

Please amend the specification as follows:

Please replace the first paragraph on page 5 with the following paragraph:

In "value prediction", an execution result of an instruction is predicted before input data necessary for the instruction is determined. In this case, output data of other instructions depending on the instruction in the program is predicted, and the instruction is executed using the predicted output data. In this way, two instructions including the data dependency are executed in parallel. As a method to predict the output data, the previous output result of the instruction is recorded, and this output data is used as the next ~~perdition~~ prediction value. Otherwise, an instruction whose output value changes by predetermined rule ([[For]] for example, the output value increases or decreases by predetermined ratio; several kinds of values are repeatedly output by predetermined order[[.]]) is found, and the output data is predicted by the predetermined rule. This "valué prediction" is studied at present. Both an applicable ratio of prediction (a ratio of the number of instructions actually applied to the prediction and the number of dynamic instructions necessary for the prediction) and a hit ratio of prediction (a ratio of the number of instructions of correct prediction and the number of instructions applied to the prediction) are not so high. A general operation unit to use this method does not exist.

a1

Please replace the first paragraph on page 15 with the following:

Hereinafter, an embodiment of the present invention will be explained by referring to the drawings. Fig. 3 is a block diagram of components of a processor (a central processing apparatus) according to the present invention. As shown in Fig. 3, the processor includes a memory 1, an address generator 2, an instruction fetch unit 3, an instruction [[cue]] queue 14, an instruction decoder 5, a task window number generator 6, an assignment unit 7, execution buffers 8, 9, 10, an operand condition decision unit 11, instruction execution units 12, 13, 14, local registers 15, 16, 17, a global register 18, a global register update unit 19, a load buffer 20, and a store buffer 21. In Fig. 3, the number of instruction execution units 12, 13, 14 (three) represents one example. The unit number is not limited to this. Furthermore, in Fig. 3, the register consists of the global register 18 accessed by all instructions and a plurality of local registers 15, 16, 17 accessed by an instruction belonging to a corresponding instruction execution unit. However, only the global register 18 may be used.

Please replace the paragraph that begins on line 19 of page 16 and ends on line 2 of page 17 with the following paragraph:

In Fig. 3, an outline of the main units in the processor is explained. The task window number generator 6 generates the task window number to assign to a decoded instruction. ~~As for at least two task windows continually located in the program, a different number is assigned to each the task window~~. A program running on the processor may contain at least two task windows. Each of the at least two task windows within the program is assigned a different number. For example, a task

a3

window number of present task window is incremented by "1" and this incremented task window number is assigned to next task window. The task window number generator 6 is realized by a loop counter whose maximum value is above "2".

Please replace the paragraph that begins on line 3 of page 17 and ends on line 17 of page 17 with the following paragraph:

The execution buffers 8, 9, and 10 are respectively connected to one of the instruction execution units 12, 13, and 14. Each decoded instruction within the program running on the processor has an address. Decoded instructions within the program are assigned to a particular execution buffer according to the decoded instruction's address. Each execution buffer ~~previously corresponds to an address of the instruction in the program and~~ consists of a plurality of ~~cues~~ queue. Each queue in the execution buffer previously corresponds to one task number. Therefore, the assignment unit 7 decides which execution buffer to assign a decoded instruction according to the address of the decoded instruction, decides the queue to insert the decoded instruction according to the task number of the decoded instruction, and inserts the decoded instruction to the queue in the execution buffer in order (first in, first out). The instructions in the plurality of queues of the execution buffer are executed in order of ~~insersion~~ insertion by the instruction execution unit.

a4

Please replace the text on page 20 (entire page) with the following text:

instruction [[cue]] queue 4 is full, the instruction fetch unit 3 stops the next fetch until the instruction queue 4 includes a space. The address generator 2 is normally updated to

a5

indicate the next address after the last fetched instruction. However, if a jump instruction is accepted or if [[a]] an lcmt instruction (loop commit instruction) is fetched, the address generator is updated to indicate the address represented by the jump instruction or the lcmt instruction. The instruction decoder 5 decodes a plurality of instructions from a head instruction stored in the instruction queue 4 through a decode bus 25 and obtains a class, an operand, a production/consumption flag, and a task number of each instruction (S12). Furthermore, the instruction decoder 5 obtains a task window number of each instruction from the task window number generator 6 and sends this information to the assignment unit 7. The assignment unit 7 selects one queue in one execution buffer into which to insert the instruction by referring to the information sent from the instruction decoder 5, and the assignment unit 7 inserts the instruction into the selected queue in the selected execution buffer through the execution buffer bus 26, 27, or 28 (S14). If the selected queue is full, the insertion waits until the selected queue includes a space. The instruction decoder 5 stops decode processing of the instruction during this period. Furthermore, if the execution buffer includes a commit

Please replace the paragraph beginning on line 4 and ending on line 16 of page 26 with the following paragraph:

On the other hand, if an instruction to output to the memory is accepted, the store buffer 21 updates a predetermined number (the number of write ports of the data cache) of values in the memory in order of registration independent of the pipeline. If an instruction for recording the class of exception is accepted, the exceptional processing

based on the class of exception is executed. An instruction to output the local register and an instruction not including output are erased from the execution buffer without update processing after acceptance/rejection of these instructions. An ~~eraser~~ erasure of the instruction in the execution buffer is realized by moving the head pointer in each [[cue]] queue.

a6

Please replace the paragraph beginning on line 20 and ending on line 25 of page 36 with the following paragraph:

As for the lcmt (loop commit) instruction, the input consists of a condition expression, a task number set consisting of a loop, an acceptable task number set for only the first loop, and a branch address if the condition expression is satisfied. The [[Lcmt]] lcmt instruction is used to represent a control dependency in a loop structure.

a7

Please replace the paragraph beginning on line 6 and ending on line 21 of page 42 with the following paragraph:

Next, at F stage in cycle 2, the instruction fetch unit 3 fetches instructions 4 ~ 6 and inserts them into the instruction [[cue]] queue 4. On the other hand, at D stage in cycle 2, the instruction decoder 5 decodes the instructions 1 ~ 3 in the instruction queue 4, and obtains a class of instruction, operand data, a task number, and a production/consumption flag. Each instruction obtains a task window number from the task window number generator 6. The assignment unit 7 respectively inserts the instructions 1 ~ 3 into a corresponding queue in the execution buffer 8 ~ 10 according to the task number. Furthermore, the assignment unit 7 sets a global register number $ 5,

a8

a local register number $ 5 of the instruction execution unit 12, and a local register number $ 1 of the instruction execution unit 13 as non-use according to the production flag and the operand of each instruction.

a8

Please replace the paragraph beginning on line 14 and ending on line 24 of page 47 with the following paragraph:

a9

In cycle 22, the instruction 16 is registered in the store buffer 21. The instruction 16 is already accepted. Accordingly, the commit bit in an entry of the instruction 16 is set to "1". On the other hand, at W stage in cycle 22, the global register $ 5 is set to "0" by the instruction 15. At E stage, the instruction 21 is executed, and tasks 2_2 and 3_1 are accepted. In this case, if the loop condition is not satisfied, tasks not accepted/rejected are erased from the execution buffer 8 ~ 10 and the instruction [[cue 3]] queue 4. Then, the address generator 2 updately indicates the next instruction 22.